bining the Power Supply scheme of the present example with schemata used for other sorts of peripheral devices, allow composition of (or build-up) a more comprehensive peripheral device schema. In turn, if the peripheral device schema is combined with a host processing support component scheme, then the result can be considered a singular, comprehensive, embedded system schema.

[0046] If (as may happen on rare occasions) one of the logical description facets can not be sensibly supported by some specific type of scheme, a facet can be stubbed with some appropriately innocuous implementation. A classic stub always succeeds as an implementation (a stub is a useful default implementation). The various schemata according to embodiments of the present invention permit some facets to be mandatory and while others are optional. In addition, default stubs are provided for most optional facets. If a given description omits them, the default applies. In many cases, this technique permits the invention to support otherwise incompatible (specific) types of power supplies. Judicious use of stubs can force a wide range of Power Supplies to meet a schemes expectations.

Fixed Executable Images & the Logical Schema

[0047] All peripheral devices may be described according to some generalized facet scheme. Each logical type of peripheral device matches one of these generalized facet templates (referred to herein as schemes). While this technique can be used in any number of contexts, and while the types of schemata used can vary as widely, each application context is best served by some correspondingly optimized collection of schemata. For instance, there is no reason to support voltage sensors in devices/products that will never include even a single voltage sensor.

[0048] As a result, it is possible to predefine a limited schema that makes provision for describing only those types of hardware peripherals that are applicable (and/or expected) for a given (application) context. One may choose to include additional schemata to increase the range of applicability (of the resulting firmware/software "part"). The expressive range of the chosen schema maps to the expected range of applicability for the corresponding embedded software/firmware/host part.

[0049] A predetermined, maximum number of logical peripheral device description instances are also employed. While the maximum number chosen is arbitrary, the present invention can leverage this chosen number of descriptions. As a practical matter, this means that different embedded software/firmware parts can reserve arbitrary—but a priori—known amounts of non-volatile (NV) storage. This fixed amount chosen tends to correspond to the maximum number of logical (and/or physical) description instances that might ever be needed (in a given application context). This maximum is preferably chosen to be large enough to cover most/all physically feasible peripheral device configurations. Although the storage size is preferably fixed, according to embodiments of the present invention storage may, alternatively, be dynamically allocated. Accordingly, the embedded firmware/software part can be either a fixed size or variable.

[0050] Those skilled in the art will realize that the maximum number of description instances permitted is often a function of costs associated with the product in which a given firmware/software part is to be embedded. While additional non-volatile storage chips can expand the operational host hardware environment (for embedded software/firmware parts), such additional chips will increase the cost.

[0051] While the present invention works with arbitrary amounts of dynamically allocated storage, those skilled in the art will realize that the present invention works when an arbitrary-but-fixed number of description entries are required. The present invention works with a tightly capped amount of non-volatile storage. If a maximum number of description instances is chosen, the layout of the embedded firmware/software's executable image does not change. A build-time option for choosing this maximum number of description entries is thus a part of embodiments of the present invention.

Free-form Description Binding

[0052] Embodiments of the present invention employ run-time binding through so-called "function pointers" in order for the embedded software/firmware code to bind to the proper logical interface for any given peripheral component. Generally the build-time compiler's calling conventions are followed, whereby each logical hardware component description supplies a handle (e.g., a memory address) to a device type/schema specific data structure. This data structure, in turn, maps to various device description-specific behaviors (e.g., function pointers).

[0053] The logical interfaces, according to embodiments of the present invention, are parameterized. This allows generalizations about a logical category of peripheral hardware components, such as Power Supplies, and permits many different ways to communicate with all of these diverse devices.

[0054] Preferred embodiments of the present invention pass a number of opaque bits as an argument/parameter to each (and all) of our logical behavior facets. The inventors have found that somewhere between 64 and 256 bits will often suffice for most arguments/parameters (if tightly packed bit fields are used aggressively). Those skilled in the art will, of course, realize that the precise number of bits depends upon the system schema in question.

[0055] This parameter passing approach allows the invention to handle both the variable numbers, and the variable types, of behavior (e.g., function/method) parameters (e.g., function/method arguments). When describing very general concepts, like power supplies, it is often the case that more specific sub-types of power supplies will need vastly different numbers, and different types, of parameters. Rather than the need to re-code/re-program, the present invention adopts a prototype based approach. All prototype behaviors need to have an interchangeable and/or standard interface.

[0056] Recall, that embedded firmware/software parts should preferably not have to be rebuilt (i.e., re-compiled/interpreted) when changes are made to the system. They must directly work with the variable numbers of interface arguments.

[0057] As an example, assume that a block of bits in a particular implementation is 64-bits. These 64 bits are typically pushed (often as machine words or dual machine words) onto the function call (stack frame) associated with each logical interface invocation (at run-time). These 64 bits